**Edge Detection – Course Project Report**

EN605.617.FA – Introduction to GPU Programming

Professor Chancellor Pascale

Miles Gapcynski

12/10/2018

Table of Contents

[Table of Figures 2](#_Toc531984023)

[Project Overview 3](#_Toc531984024)

[Introduction 3](#_Toc531984025)

[Implementation 4](#_Toc531984026)

[Convolution and Kernels 4](#_Toc531984027)

[General Approach 5](#_Toc531984028)

[CPU Approach 5](#_Toc531984029)

[GPU Approach 5](#_Toc531984030)

[Lessons Learned/Roadblocks 5](#_Toc531984031)

[Results and Analysis 6](#_Toc531984032)

[Grayscale vs HSV 6](#_Toc531984033)

[Pixel Thresholds 6](#_Toc531984034)

[Timing Results 6](#_Toc531984035)

[Conclusion 7](#_Toc531984036)

[Works Cited 8](#_Toc531984037)

# Table of Figures

[Figure 1 - Convolution (Machine Learning Guru) 4](#_Toc531984038)

[Figure 2 - Roberts Operator (Crane, 86) 5](#_Toc531984039)

[Figure 3 - Prewitt Operator (Crane, 86) 5](#_Toc531984040)

[Figure 4 - Sobel Operator (Crane, 86) 5](#_Toc531984041)

# Project Overview

## Introduction

For my course project I decided to create a program that would compare how effective the CPU and GPU are when processing image data. I’ve always been interested in the topic of image processing, but I didn’t have much experience writing software that processes image data. I ended up choosing the topic of edge detection to focus my efforts and properly scope my project for the semester. What I didn’t truly grasp until now was that edge detection itself is actually a pretty deep topic, and it has a wide range of applications it can be used for. One example that Randy Crane provides on page 79 in his book Simplified Approach to Image Processing is that edge detection can be used by photo editing software for a magic wand tool, which draws a border around pixels that match a selected pixel’s value. Crane goes on to explain that edge detection can be used in other ways, such as for identifying objects or for image registration, which “aligns two images that may have been acquired at separate times or from different sensors” (79). The effectiveness of these different algorithms/techniques depends on many factors, such as how large the image is or if the algorithm’s intention is to find the vertical edges within a picture.

The edge detection algorithm I chose to implement for my course project is to convolve different 3x3 kernels with the image data in order to identify edges within a picture. The program reads in an image specified by the user, convolves the image’s pixel data with a 3x3 kernel, and outputs the convolution results as a new image. The program was created using C++ and Cuda so that it could be run on the CPU or the GPU, allowing me to compare how effective each device is at finding the edges within a given picture.

# Implementation

## Convolution and Kernels

I chose to use convolution for my edge detection algorithm because Randy Crane points out that “common gradient (or orthogonal gradient) operators” utilize convolution to “find horizontal and vertical edges” (86). In image processing, convolution involves multiplying each pixel and it’s neighbors with the corresponding positions in the kernel, and then performing a summation of the multiplication results. The kernel represents a mask that filters the image data, producing a desired result depending on the kernel’s values (weights). The kernel is generally applied to each pixel in an iterative fashion, sliding over each row and column one by one. There are several methods for handling neighbors outside of the image bounds, such as 0 padding or wrapping, but I chose to ignore the outer pixel rows/columns for simplicity. The following diagram helps illustrate how convolution of an image and kernel works:

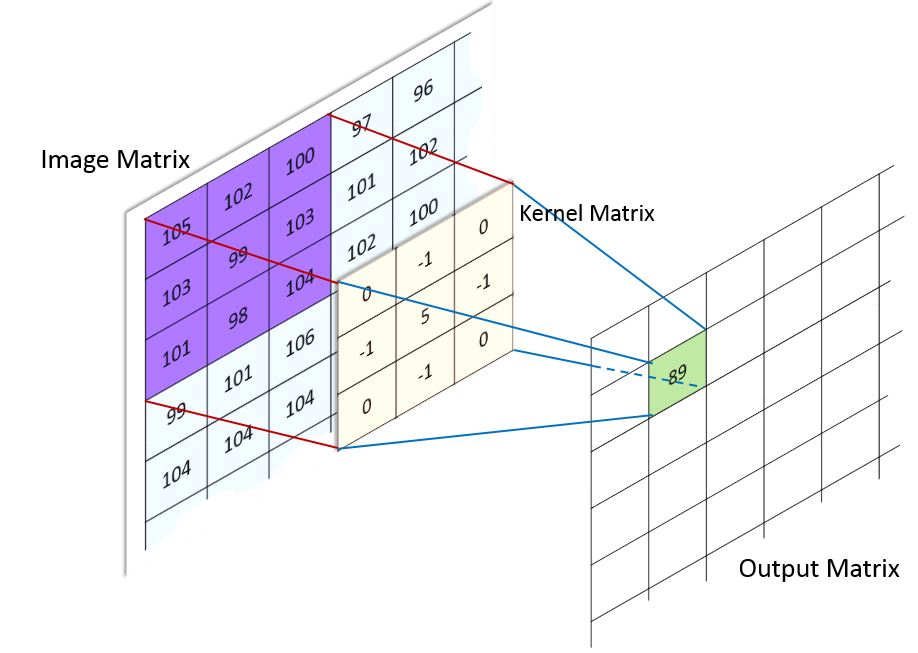


Figure - Convolution (Machine Learning Guru)

As noted earlier, Crane mentioned that common gradient operators (kernels) find both the horizontal and vertical edges within a picture by convolving each pixel position and it’s neighbors with two different kernels (86). One kernel is designed to find edges in the horizontal direction, while another kernel is designed to find edges in the vertical direction. The edge detection algorithm then calculates the magnitude of the two convolutions to produce a value representing the intensity of the edge at each pixel location. I chose three gradient operators to compare the results of: Roberts, Prewitt, and Sobel.

The Roberts operator, illustrated in figure 2, “has a smaller effective area than other masks, making it more susceptible to noise. The other masks are better able to average out fluctuations” (Crane, 86).

Figure - Roberts Operator (Crane, 86)

The Prewitt operator, illustrated in figure 3, “is more sensitive to vertical and horizontal edges than diagonal edges” (Crane, 86).

Figure - Prewitt Operator (Crane, 86)

Finally, the Sobel operator, illustrated in figure 4, “is more sensitive to diagonal edges than vertical and horizontal edges” (Crane, 86).

Figure - Sobel Operator (Crane, 86)

## General Approach

Image data can take on many different formats, and is generally comprised of a two-dimensional set of pixel data that can have 1 to N channels of values for each pixel.

* Explain approach for implementing convolution on CPU and GPU
* Grayscale and HSV

## CPU Approach

* CPU specific information

## GPU Approach

* GPU specific information (global vs shared)

## Lessons Learned/Roadblocks

* Issues ran into along the way
* If I would change anything given more time or starting over

# Results and Analysis

## Grayscale vs HSV

* Show how results differ when I convolve the kernels in grayscale vs each channel in HSV (Hue, Saturation, and Brightness/Value channels).

## Pixel Thresholds

* Show how results differ with different pixel thresholds.

## Timing Results

* Show timing results for CPU vs GPU.
* Show timing results for different sized images.
* Show timing results of GPU global vs shared memory.
* Compare cudamalloc vs cudamallochost?

# Conclusion

* Discuss how project could be expanded upon
  + It could be extended by comparing the results to other edge detection algorithms implemented in the same fashion (implemented on the CPU and GPU).
  + Another extension of this project could involve passing the resulting image to an image recognition algorithm that searches for specific objects or patterns.
  + Using texture memory
  + Using a library like NPP

# Works Cited

Crane, Randy. Simplified Approach to Image Processing

Machine Learning Guru http://machinelearninguru.com/computer\_vision/basics/convolution/image\_convolution\_1.html